Submitted and revised for the area of Software and Performance Engineering for Parallel and Distributed Systems in a special issue of PDCP (Parallel and Distributed Computing Practices. (http://orca.st.usm.edu/pdcp/) Initial version published in DAPSYS 2000.

Cluster Command & Control (C3) Tool Suite

Michael Brim, Ray Flanery, Al Geist, Brian Luethke, and Stephen Scott[D]
Computer Science & Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, TN 37830-6367 USA

Abstract

The raw computation power of today's PC clusters running the Linux operating system rivals that of commercial supercomputers at a fraction of their purchase price. As a part of that high price tag, some tools and system support come with the supercomputer "package." However, a multitude of vendors will slap together a number of commodity computer "boxes" and call it a cluster while providing nothing more than a power cord for each box. Thus, the lack of good administration and application support tools represents a hidden operation cost that is often overlooked when purchasing a cluster. This hidden cost greatly adds to the cluster's total cost of ownership. The work presented in this paper addresses the area of cluster administration and application support by presenting the Cluster Command & Control (C3) tool suite developed at Oak Ridge National Laboratory for use in administrating and using its high performance cluster computing resources.

We first cover the history of the Cluster Command and Control (C3) tool suite from initial v1.0 release to present state of v2.7 and include brief directions for v3.0 as well as the planned v4.0. We then present the requisite C3 tools environment followed by C3 command specifications with command syntax and some simple command line usage examples. A brief coverage is then give to some implementation issues between a serial and parallel implementation. We conclude the discussion of C3 v2.x with advanced usage techniques by showing how the C3 tools may be embedded into other commands and applications to create end-to-end cluster solutions. We close our discussion by providing information on C3 v3.0 that will be released later in 2001, as well as plans for a v4.0 high-scalability release.

Keywords: cluster computing, cluster tools, cluster administration, scalable computing

1 Introduction

While there are numerous tools and techniques available for the administration of clusters, few of these tools ever see the outside of their developer's cluster. Most often, they are developed for specific in-house uses. This results in a great deal of duplicated effort among cluster administrators and software developers. Thus, after building HighTORC [1], a 64-node – 128 processor – Beowulf cluster in the Summer of 1999, it was decided to make an effort to develop and collect a suite of system administration cluster tools that could be released to the cluster community so that others may benefit from our effort as well.

To simplify administration, some cluster builders simply use the Network File System (NFS) to mount one master file system on all nodes. While the "one file system" approach does greatly simplify configuration management and application development, it also provides the least scalable model and lowest performance for cluster computing due to machine and network constraints. Our approach has been to decentralize every aspect of the cluster configuration in HighTORC to promote scalability. This resulted in every node hosting its own independent operating system with the remaining local disk space used as temporary application and data storage. Of course, this approach generated the need for tools to hide the fact that HighTORC consists of 64 independent machines. Using a number of freely available tools; rsync [2], OpenSSL [3], OpenSSH [4], DHCP [5], and Systemimager [6] we developed a suite of tools collectively named the Cluster Command and Control (C3) suite. These tools are designed to easily move information into and out of the cluster as if it were a single machine rather than a group of loosely coupled independent machines. Furthermore, these tools may be used to invoke any command[1], in parallel, across the cluster. The C3 tools are similar in function to those developed by Gropp and Lusk [7] for use on MPPs – Massively Parallel Processors.

[Δ] Research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory (ORNL), managed by UT-Battelle, LLC for the U. S. Department of Energy under Contract No. DE-AC05-00OR22725.
[1] Functionally limited to non-interactive commands, as interactive commands in HighTORC's case would require the user to respond to the 64 individual node queries.

One of the main criteria of the tools is that they provide the look and feel of commands issued to a single machine. This is accomplished through using lists, or configuration files, to represent the group of machines on which a command will operate. While this requires the building of machine lists, or cluster node lists, it still presents the typical user with a single machine interface. This occurs as generally a cluster is configured and used in its entirety at any given instance by a single user. Thus the cluster node list containing all cluster nodes may be built once and then forgotten. However, a strength of the C3 tool suite is that an individual user may dynamically customize and use their cluster configuration list without affecting any other cluster users. Thus, a user may effectively partition the cluster through the use of an individualized cluster configuration node list.

A second criterion is that the tools be secure. Generally security inside a cluster, between cluster nodes, is somewhat relaxed for a number of practical reasons. Some of these include improved performance, ease of programming, and all nodes are generally compromised if one cluster node’s security is compromised. Because of this last issue, security from outside the cluster into the cluster is of utmost concern. Therefore, user authentication from outside to inside must be done in a secure manner.

The third criterion is tool scalability. A user may tolerate an inefficient tool that takes a few minutes to perform an operation across a small cluster of 8 machines as it is faster than performing the operation manually 8 times. However, that user will most likely find it intolerable to wait over an hour for the same operation to take effect across 128 cluster nodes. Further complicating matters is that many cluster sites are now hosting multiple clusters that are or will eventually be combined into federated computation clusters, also called clusters-of-clusters. Extending this paradigm even further is the computation Grid [8] where combining federated clusters from multiple sites is the norm.

Toward this effort, this paper describes the command line interface tools for Cluster Command and Control (C3) developed for use on HighTORC. We first cover the history of C3 from initial v1.0 release to present state of v2.7 and include brief directions for v3.0 as well as the planned v4.0. We then present the requisite C3 tools environment followed by C3 command specifications with command syntax and some simple command line usage examples. A brief coverage is then give to some implementation issues between a serial and parallel implementation. We conclude the discussion of C3 v2.x with advanced usage techniques by showing how the C3 tools may be embedded into other commands and applications to create end-to-end cluster solutions. We close our discussion by providing information on C3 v3.0 that will be released later in 2001, as well as plans for a v4.0 high-scalability release.

2 C3 Development History

The idea for a suite of cluster tools that would treat the cluster as one system was born out of need. The 64-node cluster HighTORC was delivered to Oak Ridge National Laboratory in mid-summer 1999. Throughout that Summer and Fall we worked “harder” by manually installing and configuring the cluster and all the necessary application and computing environment tools. When something necessitated a change to the cluster configuration, be it a failure of hardware or software, the addition of a new library, a patch to an existing application, or even the expansion of our user base, everything came to a halt while we manually modified, moved, and tested the new configuration. We quickly realized that if we ever wanted to do anything other than administer our cluster, with more clusters and users on the horizon that we had to begin working “smarter.” Thus was born the Laboratory Directed Research Development (LDRD) project in Fall 1999 that resulted in the development of the Cluster Command and Control (C3) tools suite.

February 2000 was the internal release date of C3 version 1.0, a proof-of-concept release that followed the serial execution model where each command was initiated in succession across the cluster nodes. Here, the initiating node, or head node, would initiate a command and wait for each node to return its status or result prior to initiating the command on the next node. This resulted in a sequential execution time across the cluster represented by the lengthy equation of roughly – cluster command execution time = number of nodes * (communication time + startup time + operation time + return result time).Version 1.x had other shortfalls including that some commands were too tightly coupled to the HighTORC cluster environment. Thus, while version 1.x was a great success in providing assistance to ORNL in the administration and operation of HighTORC, it failed as a general-purpose cluster tool. Thus was laid the groundwork for a version 2.x universal release.

Version 2.0 was completed in May 2000. This version was successful in removing ORNL cluster dependencies, paving the way for a generalized C3 tool suite. This version also saw the initial attempt to initiate parallel command execution across all cluster nodes. Parallel execution was done with the Perl threads package in Perl 5.6. Unfortunately it was discovered that this version of the Perl threads package was extremely unstable. Symptoms first showed up as a number of cluster nodes hanging or dropping out of a C3 command. While inconsistent, at times ten to twenty percent of the nodes would fail to complete a task as simple as a directory listing. Thus, version 2.0 was never released to the public and immediate work began on another parallel execution technique.

While there were numerous iterations between version 2.0 and 2.7, the next major release of C3 was version 2.6 in September 2000. This was also the first widely publicized C3 release made available on the web. This version employed a multi-processing implementation. Starting at version 2.5, C3 has proven itself to be a very stable and robust general-purpose cluster tool. Version 2.6 has also been included as part of the Open Cluster Group's [9] – developer's release of the Open Source Cluster Application Resources [10] (OSCAR) package. C3 version 2.7 will be included in the general public release v1.x of OSCAR.

At this writing, the most recent C3 version publicly available is version 2.7. This version has an updated application program interface (API) and follows a more uniform naming convention. We will expand on the future plans for C3 in a later section of this paper. However, to complete the current history of C3, it is sufficient to say here that version 3.0 is presently under development with an expected release of Summer 2001 and a version 4.0 is in the design phase with an anticipated beta release in early 2002.

3 Requisite Software Environment

A number of tools developed and freely published on the web were collected to facilitate the development of the C3 tool suite. The following section briefly describes each requisite software package and provides information where they may be obtained. Detailed information on each tool should be obtained from their respective developer site as indicated in the appropriate section.

3.1 rsync

Rsync is an efficient method for mirroring files between two machines similar to rdist [11]. Both tools operate by checking for differences between the files on the source machine and those on the destination. Rdist simply checks the timestamp on each file to see if it needs updated, and sends the entire updated file from source to destination when necessary. Rsync, on the other hand, uses an advanced algorithm for checking to see if files need to be updated and to perform subsequent updates. Briefly, rsync operates by:

1. A user specifies the source and destination machines at the command line, as well as the file to update.
2. The destination machine splits the target file into small fixed-size blocks, and generates two checksums for each block – a weak "rolling" 32-bit checksum and a strong 128-bit MD4 checksum.
3. The destination machine transfers its checksums to the source.
4. The source machine searches through its local file to find all blocks for which the destination's checksums match its own.
5. The source machine generates a sequence of instructions that the destination can use to update the file.
6. The source machine then sends the instructions, along with any new data, to the destination.
7. The destination machine uses the instructions and new data to reconstruct its files to the newly updated state.

A more detailed explanation of the rsync algorithm may be found in [12]. As a result of using the above algorithm, rsync is able to greatly reduce network traffic by sending only new data to the destination node, along with a small amount of overhead.

In a typical cluster, the files on the server will be very similar to those on the client nodes. To take advantage of this property, rsync is used in C3 commands that effect the movement of files between server and nodes. In addition, the system-imaging tool Systemimager, described in section 3.4, also uses rsync in performing efficient system image updates.

3.2 Authentication and data transport options

Both the secure shell (ssh) and the remote shell (rsh) may be used for user authentication and data transport within C3. While rsh is a very simple and direct technique, using rsh as the root user represents a very large security risk. Thus, when using the C3 tools as root, ssh is the preferred solution as it offers a more secure authentication approach. As such, the C3 tools use ssh by default. However, the tools can be forced to use rsh by setting the environment variable `C3_RSH` to '1'.

In the bash shell, the environment variable `C3_RSH` is set by typing the following on the command line or inserting it in the `$HOME/.bashrc` file, which will set it whenever the user starts the shell.

```
export C3_RSH=1
```

For the (t)csh shell, the environment variable `C3_RSH` environment variable can be set at the command line or in `$HOME/.cshrc` using the following command.

```
setenv C3_RSH 1
```

3.2.1 rsh

Rsh, or remote shell, is a standard Unix utility that allows remote command execution. Rsh's authentication is accomplished by either a global set of files (`/etc/hosts.allow`, `/etc/hosts.equiv`, `/etc/hosts.deny`) or a local file (`$HOME/.rhosts`). This has been shown to be relatively insecure, due to the fact that these files are globally readable and easily changed by malicious users. However, on a private network, such as those where many clusters are built, the overhead and complications associated with ssh may not be necessary or desirable, other than in the case of root access, so the ability to use rsh for authentication was included.

3.2.2 OpenSSH

OpenSSH is an open source tool that implements the secure shell 1.5 and 2.0 protocols. OpenSSH requires the OpenSSL encryption libraries to work. OpenSSH uses a digital signature generated by OpenSSL for authentication purposes. This eliminates the need to send passwords across the network to remote machines. This also allows a reliable and secure method, as opposed to rsh, for authenticating root. OpenSSH is an enabling tool that provides a secure means for C3 to connect to cluster nodes from the cluster's head node. Without such security the C3 tool suite would either not be permitted to execute or at the very least would be restricted by most site security policies. C3 uses OpenSSH for a secure means to allow the root user to login to each cluster node when invoking commands and transferring files.

3.3 DHCP

Dynamic Hosts Configuration Protocol (DHCP), is used to allocate IP addresses to machines on a given network. It can do this dynamically within a given range of IP addresses or statically by associating a NIC's MAC address with a specific IP address. Dynamic allocation of IP addresses makes it easy to swap out machines with little operator intervention. Static allocation of IP address makes it easier to troubleshoot machine/network problems and subsequently debug distributed cluster codes.

Dynamic IP allocation allows DHCP to "lease" an IP address to a client for a specified time period. This time period may be set from a few seconds to forever. "Forever" in this case ends when the client gives up the IP address – like on a reboot. A short lease will be detrimental to cluster performance, as it will require the cluster nodes to continually access the DHCP server to obtain a new IP address. A forever lease will produce stability at least until a node crashes or reboots. However, periods of instability in clusters tend to affect more than one node thus making debugging using the IP address useless, as they will shuffle on many nodes simultaneously. Our recommendation is to initially dynamically allocate IP addresses within a specified range in order of cluster node identifier (number). Then use the Systemimager tool, described below, to change the dynamic allocation to a static allocation scheme. Thus, ease of initial installation with troubleshooting and debugging capabilities is retained. In practice, the static allocation of IP address on clusters eases the replacement of a cluster node. This occurs as generally when a cluster node fails, another machine (node) is placed into that physical slot effectively assuming the identity of the replaced machine. If many simultaneous node failures occur, as in the case of a non-protected power surge, it is actually simpler to physically replace failed machines and then go through the startup/dynamic allocation and convert to static allocation, as was the case of initial cluster setup.

3.4 Systemimager

Systemimager is a cluster system administrator tool, run by the root user, which enables a cluster node to *pull* the current cluster node image from an image server. This proves to be very valuable when initially installing the cluster environment and for subsequent cluster wide operating system and software environment updates. The current version of Systemimager at this writing is v1.4.

When combined with network booting, Systemimager provides a simplified technique to place the initial cluster image on each cluster node. As each machine is network booted, it will *pull* the current image from the image server. The image is downloaded, installed, and now the machine may reboot as a cluster node. The only task remaining is to assign a network IP address to the node.

Systemimager requires DHCP to be installed and run on the cluster image server. It is also a good idea to do this for cluster computing in general. The nodes being updated can either be configured using dynamically or statically assigned IP addresses. DHCP dynamically assigns IP addresses by default as machines boot. However, statically assigned addresses are generally advocated for cluster use as it facilitates node debugging and troubleshooting. However, one cannot store statically assigned IP addresses on a cluster image when pushing the same image across a number of machines. To do so would produce disastrous results, as all machines would receive and then try to use the same IP address. The alternative is to manually change the IP address of each machine after initial startup and then reboot with the correct IP address. This is not very practical as it is time consuming, error prone, and becomes rather annoying after about four machines. Systemimager provides a solution to this problem via it's `makedhcpstatic` utility. To use the utility, simply boot cluster nodes in order of desired IP address, as DHCP will sequentially assign the IP number to each machine as it comes online. Next, run `makedhcpstatic`, which will rewrite the DHCP configuration file (`/etc/dhcpd.conf`) to associate each node's MAC address with its hostname and restart DHCP. Now each time a node requests an IP address, the same number will be assigned. This technique works great for installing a large number of machines with contiguous IP addresses. However, if a machine or NIC is replaced, you must manually set the IP address on the node and in the DHCP configuration file. However, this is a small price to pay for the occasional machine or NIC failure compared to the effort required to initially build and configure a large cluster.

One of the shortcomings of Systemimager is that it requires a cluster node to request an image from the cluster image server. While this technique avoids the security problems of an outside machine forcing new software on a cluster node and perhaps taking control of the machine, it also restricts the usefulness of the image update to preplanned updates (*pulls*) driven by coordinated node cron jobs. Our desire was to augment this feature of Systemimager such that a system administrator may effectively orchestrate the *push* of a new system image in a secure manner across the entire cluster or any portion of the cluster. To implement such functionality, the C3 command `cpushimage` was created. This command is described in detail in the following section.

4 C3 Tools Suite

Eight general use tools have been developed in the effort thus far: `cexec`, `cget`, `ckill`, `cps`, `cpush`, `cpushimage`, `crm`, and `cshutdown`. **`cexec`** is the C3 general utility in that it enables the execution of any standard command across a cluster. **`cget`** retrieves files from cluster nodes into a user specified location on the local machine. **`ckill`** is used to terminate a given process across the cluster. **`cps`** returns the aggregate result of the 'ps' command run on each cluster node. **`cpush`** will let you push individual files or directories across the cluster. **`cpushimage`** is our cluster-wide image *push* answer to the single machine Systemimager *pull* solution. **`crm`** permits the deletion of files or directories across the cluster. **`cshutdown`** can be used to shutdown or reboot cluster nodes. `cpushimage` and `cshutdown` are both system administrator tools that may only be used by the root user. The other six tools may be employed by any cluster user for both system and application level use.

For each tool, there are two execution methodologies available to users. The default method of execution is parallel, where the command is run on all specified nodes concurrently. The other method of execution is serial, which will iteratively run through each node specified. Using the command name as shown will use the parallel version, whereas adding an 's' to the end of the name specifies the serial version. For example, `cexec` is the parallel version, while `cexecs` is serial.

All of the tools use a cluster configuration file to determine the cluster nodes on which commands should be run. The configuration file format is one node specifier per line, where the node specifier can be an IP address or a hostname that can be resolved using standard methods. *The cluster's head node should not be listed in the configuration file*, only the client machines. It is possible to destroy, via overwriting the head node if it is included as a computation node in the cluster configuration file. By default, the tools use the configuration file `/etc/c3.conf`, which contains all the client nodes. Each tool also has an optional command line option that can be used to specify an alternate cluster configuration file. An example configuration file for a cluster with four client nodes is shown below.

```
node1.torc.ornl.gov
node2.torc.ornl.gov
node3.torc.ornl.gov
node4.torc.ornl.gov
```

4.1 cexec

The `cexec` command is the general utility tool of the C3 suite in that it enables the execution of any command on each cluster node. As such, `cexec` may be considered the clusterized version of `rsh/ssh`. A command string passed to `cexec` is executed "as is" on each node. This provides a great deal of flexibility in both displaying the command output and arguments passed in to each instruction. A trace of the actions of `cexec` is given in Figure 1.

SYNOPSIS

```
cexec[s] [OPTIONS] --command="command string"
```

OPTIONS

`-c, --command "command string"`	: command string to be executed on each node
`-h, --help`	: display help message
`-l, --list nodelist`	: alternate cluster configuration file
`-p`	: print node name with output (serial only)

GENERAL

There are two basic ways to call `cexec`:

1. To execute a command (in parallel by default)

   ```
   cexec --command="mkdir temp"
   ```

   Notice the use of the quotation marks, allowing Perl to interpret what is inside the quotes as a single string.

2. To print the node name and then execute the string

   ```
   cexecs -p -c "ls -l"
   ```

   This form of the command allows the ability to read the output from a command such as 'ls' and to know which machine the message came from.

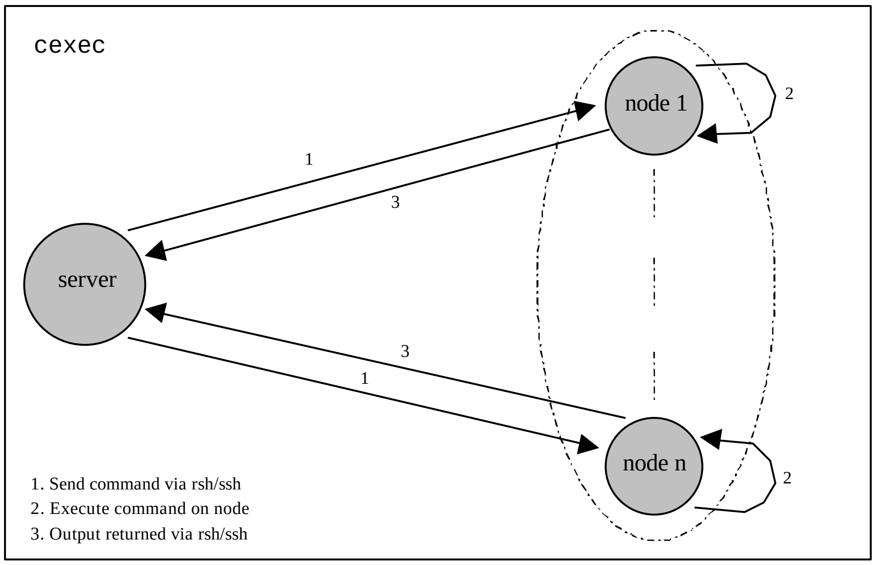
cexec
server
node 1
node n
1
2
3
1. Send command via rsh/ssh
2. Execute command on node
3. Output returned via rsh/ssh

Figure 1 – The cexec command operation.

4.2 cget

The `cget` command, whose trace is given in Figure 2, will retrieve the given files from each cluster node and deposit them in a specified directory location on the local machine. Since all files will originally have the same name, only from different nodes, an underscore and the node's IP or hostname is appended to each file name. Whether the IP or hostname is appended depends on which is specified in the cluster specification file. Note that `cget` operates only on files and ignores subdirectories and links.

SYNOPSIS

```
cget[s] [OPTIONS] --source=pattern --target=location
```

OPTIONS

`-e, --server` *`hostname`*	: allows explicit naming of the server, useful when the server has both internal and external network connections – since *cget* uses a get_hostname function, the wrong name may be returned for the internal network
`-h, --help`	: display help message
`-l, --list` *`nodelist`*	: alternate cluster configuration file
`-s, --source` *`pattern`*	: pattern or file to get
`-t, --target` *`location`*	: location to put files on local machine

GENERAL

There are two basic ways to call `cget`:

1. To get a given pattern (in this case a whole directory)

   ```
   cget --source=/home/usr/\* --target=/home/usr/
   ```

   Notice the use of a '\' before the special character '*'. The shell tries to expand wildcards before the program is called and this forces the shell not to expand them.

2. To get a single file

   ```
   cget –s /home/usr/filename –t /home/
   ```

Notice that the target is always a directory, as this is the destination for files retrieved.

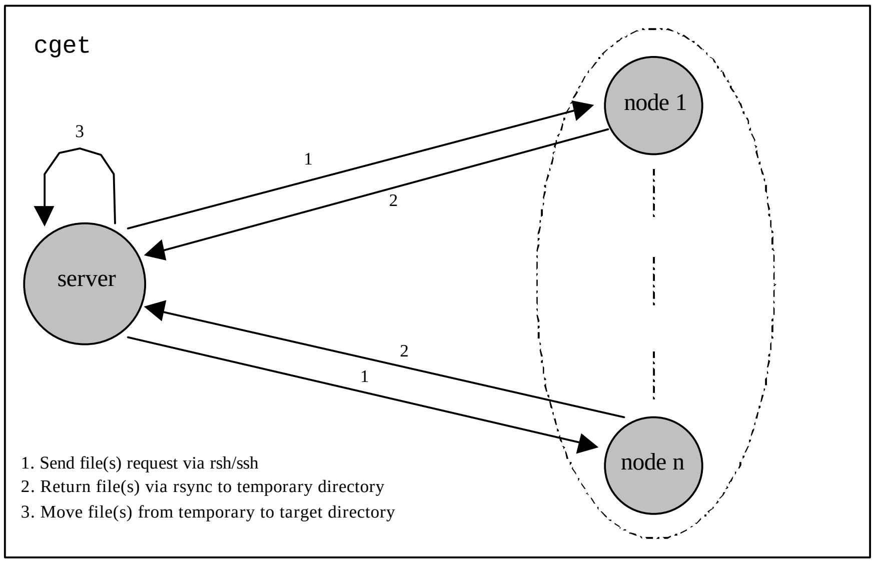
cget
node 1
node n
server
1
2
3
1. Send file(s) request via rsh/ssh
2. Return file(s) via rsync to temporary directory
3. Move file(s) from temporary to target directory

Figure 2 – The `cget` command operation.

4.3 ckill

The `ckill` tool runs the standard Linux '`kill`' command on each of the cluster nodes for a specified process name. Figure 3 shows a trace of the `ckill` operation. Unlike '`kill`', `ckill` must use the process name as the process ID (PID) will most likely be different on the various cluster nodes. The root user has the ability to further indicate a specific user in addition to process name. This enables root to kill a specific user's process by name and not affect other processes with the same name but owned by other users. Root may also use signals to effectively do a broad based kill command.

SYNOPSIS

`ckill[s] [OPTIONS] --signal=`*`signal`*` --process=`*`process-name`*

OPTIONS

`-h, --help`	: display help message
`-l, --list` *`nodelist`*	: alternate cluster configuration file
`-p, --process` *`process-name`*	: the name of the process being killed (not PID or job number)
`-s, --signal` *`signal`*	: use the same signals you would normally use with 'kill'
`-u, --user` *`username`*	: the user name of the process owner, or 'ALL' to specify all users - searches `/etc/passwd` for a UID. (root user only)

GENERAL

An example usage of `ckill` is as follows:

```
ckill --signals=9 --process=a.out --user=ALL
```

Does a '`kill –9`' (unconditional `kill`) on all `a.out's`, regardless of process owner.

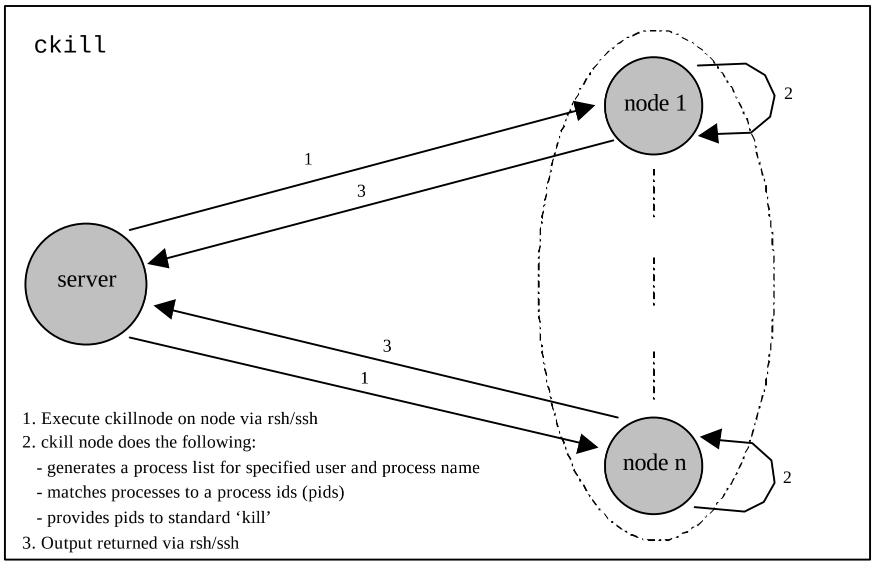
ckill
node 1
server
node n
1
3
2
3
1
2
1. Execute ckillnode on node via rsh/ssh
2. ckill node does the following:
- generates a process list for specified user and process name
- matches processes to a process ids (pids)
- provides pids to standard 'kill'
3. Output returned via rsh/ssh

Figure 3 – The `ckill` command operation.

4.4 cps

The `cps` tool runs the standard '`ps`' command on each node of the cluster with the options specified by the user. Figure 4 shows a trace of the `cps` operation. The output for each node is stored in `$HOME/ps_output_hostid`, where `hostid` is the host identifier (name or IP address) used in the cluster configuration file.

SYNOPSIS

`cps[s] [OPTIONS] --options=`*`ps-options`*

OPTIONS

`-e, --server` *`hostname`*	: allows explicit naming of the server, useful when the server has both internal and external network connections – since *cps* uses a get_hostname function, the wrong name may be returned for the internal network
`-h, --help`	: display help message
`-o, --options` *`ps-options`*	: the options you want '`ps`' to use, any options '`ps`' recognizes are fine
`-l, --list` *`nodelist`*	: alternate cluster configuration file

GENERAL

An example usage of `cps` is as follows:

`cps --options=A`

Runs '`ps`' with the `-A` option on all nodes. If more than one option is needed, use "`--options=aux`" for the '`a`', '`u`', and '`x`' options.

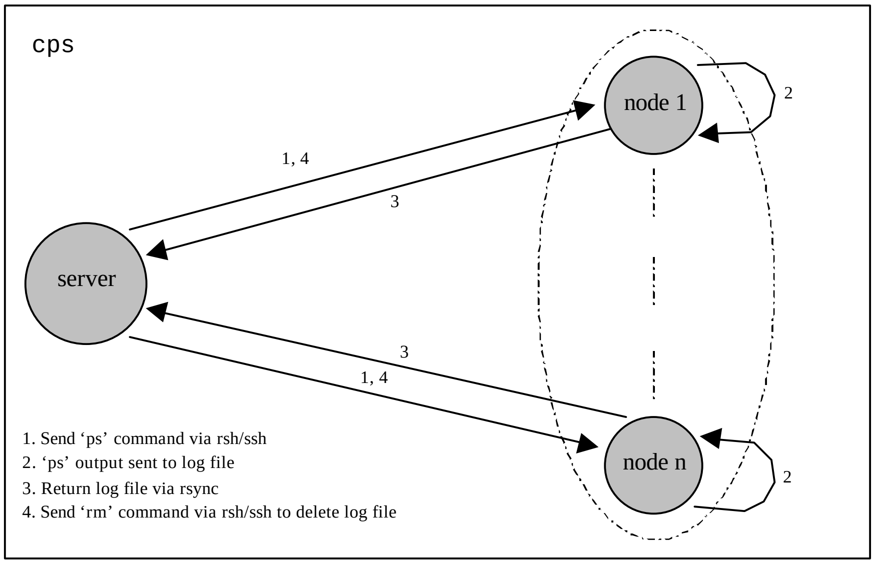
cps
node 1
server
node n
1, 4
3
2
3
1, 4
2
1. Send 'ps' command via rsh/ssh
2. 'ps' output sent to log file
3. Return log file via rsync
4. Send 'rm' command via rsh/ssh to delete log file

Figure 4 – The cps command operation.

4.5 cpush

While `cpushimage` has the ability to push an entire disk image to a cluster node, as an application support tool, it is too cumbersome when one simply desires to push files or directories across the cluster. Furthermore, `cpushimage` is only available to system administrators with root level access. From these restrictions grew the desire for a simplified cluster push tool, `cpush`, providing the ability for any user to push files and entire directories across cluster nodes. As shown in Figure 5, `cpush` uses `rsync` to push files from server to cluster node.

Caution **–** do not use `cpush` to push the root file system across nodes. Systemimager provides a number of special operations to enable cpushimage and `updateimage` to properly perform this task.

SYNOPSIS

`cpush[s] [OPTIONS] --source=`*`pattern`*` --destination=`*`location`*

OPTIONS

`-d, --delete`	: removes any files on the nodes not on the source machine
`-e, --destination` *`location`*	: the destination file or directory on the nodes
`-f, --file` *`filelist`*	: a list of files to be pushed (must be used with -e option)
`-h, --help`	: display help message
`-l, --list` *`nodelist`*	: alternate cluster configuration file
`-s, --source` *`pattern`*	: file or pattern to push
`-x, --exclude` *`pattern`*	: file or pattern to exclude from push

GENERAL

There are several different ways to call `cpush`:

1. To move a whole directory

```
cpush --source=/home/\* --destination=/home/
```

The use of the backslash before the '*' prevents the shell from trying to expand the special character before the call is made.

2. To move a single file

```
cpush --source=/home/filename --destination=/home/
```

3. To move a single file, renaming that file on the cluster nodes

```
cpush --source=/home/filename1 --destination=/home/filename2
```

4. To move a set of files matching a pattern

```
cpush --source=/home/\*.\*c --destination=/home/
```

   Again, notice the backslashes preceding the special characters to prevent the shell from expanding them.

5. To move a set of files listed in a file

```
cpush –-file=/home/filelist --destination=/home/
```

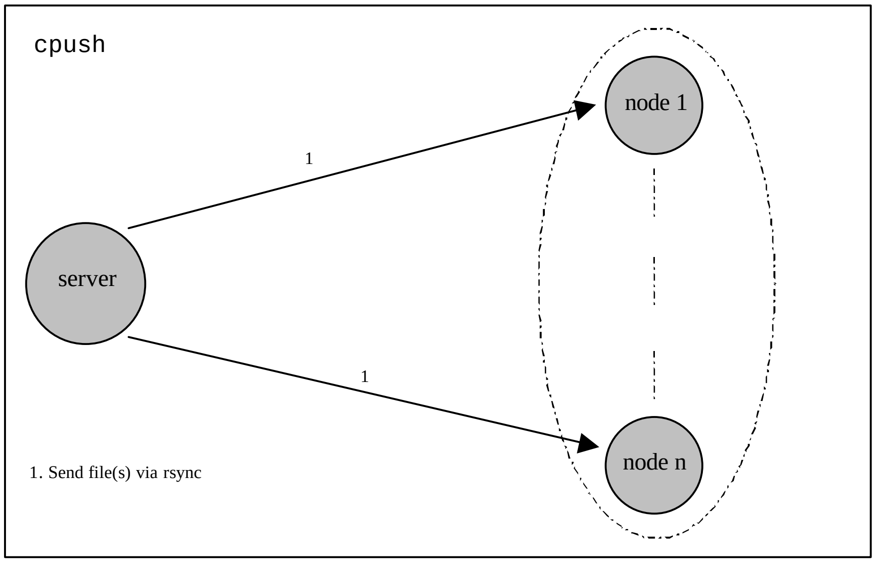

Figure 5 – The `cpush` command operation.

4.6 cpushimage

`cpushimage` enables a system administrator logged in as root to push a cluster node image across a specified set of cluster nodes and optionally reboot those systems. This tool is built upon and leverages the capabilities of Systemimager. While Systemimager provides much of the functionality in this area, it fell short in that it did not enable a cluster-wide *push* for image transfer. `cpushimage` essentially pushes a request to each participating cluster node to pull an image from the image server. Each node then invokes the pull of the image from the cluster image server. Of course, this description assumes that Systemimager has already been employed to capture and store a cluster node image on the cluster image server machine. A trace of the actions of `cpushimage` is shown in Figure 6.

SYNOPSIS

`cpushimage[s] [OPTIONS] --image=`*`imagename`*

OPTIONS

`-e, --server` *`hostname`*	: allows explicit naming of the image server, useful when the local machine has both internal and external network connections – since `cpushimage` uses a `get_hostname` function, the wrong name may be returned for the internal network

`-h, --help`	: display help message
`-i, --image` *`imagename`*	: name of system image
`-l, --list` *`nodelist`*	: alternate cluster configuration file
`--nolilo`	: don't run LILO after update
`-r, --reboot`	: reboot nodes after updates complete

GENERAL

An example usage of `cpushimage` is as follows:

```
cpushimage –r ––image=myimage
```

Updates the system image on all nodes using the image '`myimage`,' rebooting each node as it finishes.

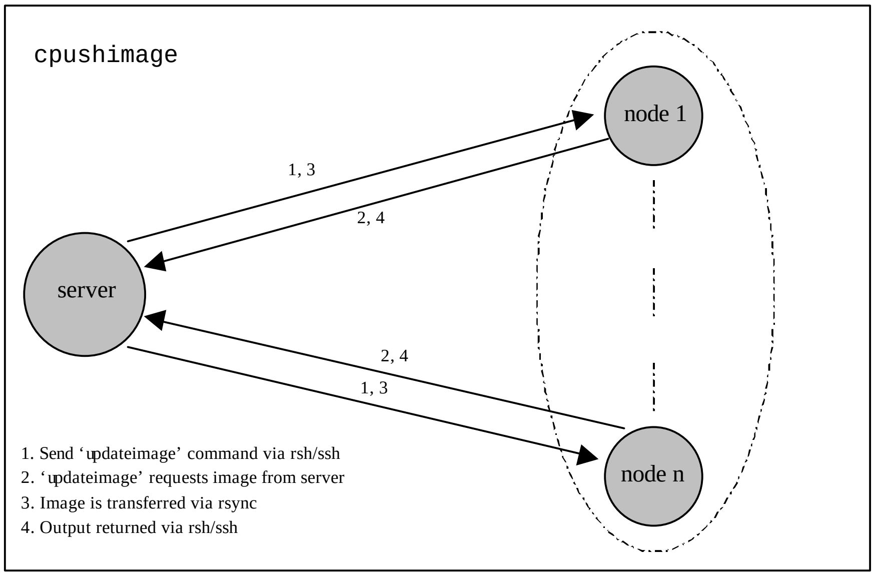
cpushimage
server
node 1
node n
1, 3
2, 4
2, 4
1, 3
1. Send 'updateimage' command via rsh/ssh
2. 'updateimage' requests image from server
3. Image is transferred via rsync
4. Output returned via rsh/ssh

Figure 6 – The `cpushimage` command operation.

4.7 crm

`crm` is a clusterized version of the standard '`rm`' delete file/directory command. Figure 7 shows a trace of the `crm` operation. The command will go out across the cluster and attempt to delete the file(s) or directory target in a given location across all specified cluster nodes. By default, no error is returned in the case of not finding the target. The interactive mode of '`rm`' is not supplied in `crm` due to the potential problems associated with numerous nodes asking for delete confirmation.

SYNOPSIS

```
crm[s] [OPTIONS] --files=pattern
```

OPTIONS

`-f, --files` *`pattern`*	: file or pattern to delete
`-h, --help`	: display help message
`-l, --list` *`nodelist`*	: alternate cluster configuration file
`-r`	: recursive delete
`-v`	: verbose mode, shows error message from 'rm'

GENERAL

There are several different ways to call `crm`:

1. To delete a directory (must be done recursively)

```
crm –r --files=/home/usr/\*
```

Notice the use of a '\' before the special character '*'. The shell tries to expand wildcards before the program is called, and this forces the shell not to expand them.

2. To delete a single file

```
crm –f /home/filename
```

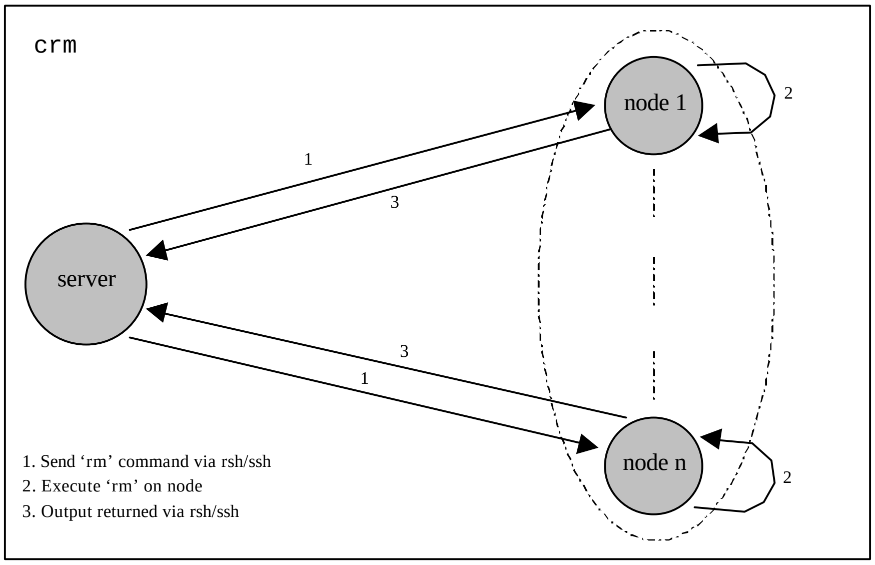
crm
server
node 1
node n
1
2
3
1. Send 'rm' command via rsh/ssh
2. Execute 'rm' on node
3. Output returned via rsh/ssh

Figure 7 – The `crm` command operation.

4.8 cshutdown

Without a cluster shutdown command it is very time consuming to log onto each node and perform an orderly shutdown process. If the direct approach of simply powering down machines is taken, the penalty will be paid on the subsequent startup as each machine will then spend time checking its respective file system. Although most clusters are not frequently shutdown in their entirety, clusters that multi-boot various operating systems will most definitely benefit from such a command, as will all clusters after updating the operating system kernel. Also, on those rare occasions where a cluster must be brought down quickly, such as when on auxiliary power due to a power outage, `cshutdown` is much appreciated. Thus, `cshutdown` was developed to avoid the problem of manually talking to each of the cluster nodes during a shutdown process. As an added benefit, many motherboards now support an automatic power down after a halt, resulting in an "issue one command and walk away" administration for cluster shutdown. Figure 8 shows a trace of the `cshutdown` operation.

SYNOPSIS

`cshutdown[s] [OPTIONS] --options=`*`options`*` -t=`*`time`*

OPTIONS

`-h, --help`	: display help message
`-l, --list` *`nodelist`*	: alternate cluster configuration file
`-m, --message` *`"message"`*	: message to display on each node
`-o, --options` *`options`*	: the options you want 'shutdown' to use, any options '`shutdown`' recognizes are fine
`-r, --onreboot` *`label`*	: use LILO label upon reboot
`-t` *`minutes`*	: time before shutting down

GENERAL

An example usage of `cshutdown` is as follows:

```
cshutdown --options=h –t=1 --message=”going down for maintenance”
```

Halts each machine in one minute displaying the given message on each node.

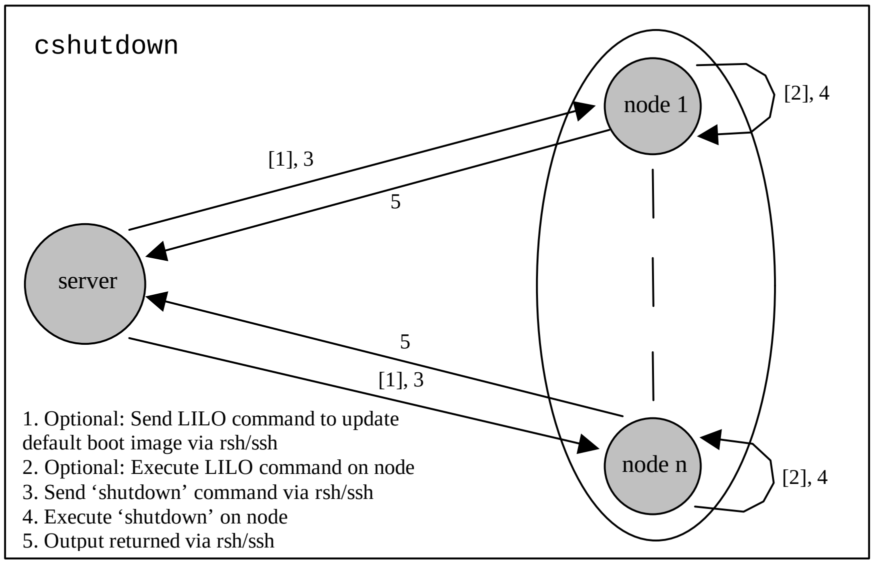
cshutdown
node 1
[2], 4
[1], 3
5
server
5
[1], 3
node n
[2], 4
1. Optional: Send LILO command to update default boot image via rsh/ssh
2. Optional: Execute LILO command on node
3. Send 'shutdown' command via rsh/ssh
4. Execute 'shutdown' on node
5. Output returned via rsh/ssh

Figure 8 – The `cshutdown` command operation.

5 Serial Vs Parallel C3 Tool Execution

While the serial version accomplished many of our initial goals it was severely lacking in scalability. One of the solutions to fix this was to make a parallel version. The first parallel version used the Perl threads package included in Perl 5.6 as a compiled module. However, the implementation of the Perl threads suffered from a severe problem with race conditions. On our cluster of 64 machines, we would typically have failure rates of ten to twelve nodes when attempting to execute multiple threads. The only way to end a command was to manually kill each process, attempt to determine which machines had failed, and then manually fix them. This failure rate was of course unacceptable. As a result, the next version of C3 was implemented using multiple concurrent processes – forking a process for each node. Forking proved to be a much more robust solution than using Perl threads, as node failures are now only typically caused by hardware failures.

While administrating the HighTORC cluster, we found the speed increase associated with the parallel version of the tools to be very nice. However, due to the indeterminate nature of the multi-process version the serial version is still included. The serial version of each tool is very useful for debugging system services, as each machine is processed according to its order in the configuration file, allowing you to see on which nodes the command completed successfully. The serial version of `cexec` is also useful as a quick method of verifying network connectivity to each node in the cluster.

6 Advanced Usage

In addition to using the tools interactively on the command line, the tools can also be used in shell scripts just like any other standard command. By adding C3 commands to scripts, users and administrators can perform advanced tasks that had to be done manually in the past.

One example of an advanced task that system administrators may employ is scripted software installations on all cluster nodes. With the C3 tools, administrators have many options as to how they could complete this task. One method would be to update one node, create a new system image from the updated node, and then push the image across the cluster using `cpushimage`. However, if the software installation is a small task, administrators may not want to go to the trouble of generating a new image and pushing it across the cluster. In this case, the administrator could use a script such as in Figure 9 to accomplish the task.

```
#!/bin/sh

# copy over package tarball
/usr/bin/cpush -s /root/software/mpich.tar.gz -e /tmp

# unpack tarball (creates mpich subdirectory)
/usr/bin/cexec -c "tar -zxf /tmp/mpich.tar.gz”

# build & install
/usr/bin/cexec -c "cd /tmp/mpich; ./configure -prefix=/usr/local/mpi"
/usr/bin/cexec -c "cd /tmp/mpich; make; make install"
```

Figure 9: System Administrator Script

In the script shown in Figure 9, the administrator is installing a package from a tarball by pushing the tarball to all nodes using `cpush`, unpacking the tarball on all nodes using `cexec`, and then running the commands necessary to build and install the package using `cexec`. Alternatively, the administrator could have just created a script to install the package on a local machine, pushed the install script and tarball to all nodes, and then run the install script on all nodes using `cexec`. This is the method used for some of the software installation tasks within the Open Source Cluster Application Resources (OSCAR) project. C3 is also included as a system administration and programming environment tool in OSCAR.

For general users, scripting the C3 tools can also be advantageous. A common advanced task that users may use the tools for is running a parallel job and collecting the output files generated on all of the nodes. Figure 10 shows a script in which a user pushes his executable to a temporary directory on all nodes, runs a parallel job, and then retrieves the results to a local directory.

```
#!/bin/sh

uid=`id -u`
user=`id -un`
dir="/tmp/myapp.$uid"
app="/home/$user/apps/myapp/hello"
results="/home/$user/apps/myapp/results"

# create temporary directory
/usr/bin/cexec --command="mkdir $dir"

# copy over application binary (hello)
/usr/bin/cpush --source=$app --destination=$dir

# run hello (which creates a hello.out on each node)
/usr/local/mpi/bin/mpirun -np 64 $dir/hello

# collect output files
/usr/bin/cget --source=$dir/hello.out --target=$results

# remove temporary directory & contents
/usr/bin/crm -r -f $dir
```

Figure 10: User Application Script

Another embedded use for C3 has been in the Managing and Monitoring Multiple Clusters (M3C) [13] project. Here, the C3 commands are invoked from within the M3C Java development environment as plug-in cluster administration / operation tools. Briefly, M3C was developed to provide a framework with an easy to use web-based graphical user interface that may "plug-in" multiple tools to provide a variety of solutions for dealing with the unique challenges of cluster administration and operation. It is designed as an extendable framework that can work with different underlying back-end tools. The M3C framework is a web-based system designed such that it will operate on multiple clusters as transparent as it does on a single cluster. Furthermore, since M3C is web-based, users may interact with participating clusters via remote Internet access. Thus, users may easily share their computational resources with others via the web-based interface.

7 Future plans for the C3 tools suite

At the time of writing this document, the C3 tool suite has been in development for slightly over 1-year. In this time we have learned many lessons and evolved the existing tools to better handle cluster administration as well as general user needs. However, we presently feel that we have taken the current tool suite as far as it can go within the constraints of its underlying implementation. Thus we started this year with a self-given directive to take our "lessons-learned" thus far and start over with a clean slate. The only self-imposed constraint is that the new version loses no functionality.

While wanting to serve the general cluster community, we also find ourselves guided by an internal directive that the development of new cluster tools should look toward supporting extremely large-scale clusters. Thus, our future work is concurrently following two paths with a planned convergence in the end. In one path is version 3.x that will provide additional growth in the following areas: better management of multiple clusters – in particular, those clusters in multiple administrative domains with computation nodes on a private network – not exposed to direct outside access; enhanced API with support for user specified node ranges; and because of some limitations with Perl and features of Python, this implementation will be in Python. On the other path is version 4.x – the first C3 "scalability" release. This version will differ greatly in implementation, capability, and capacity when compared to prior versions. While we intend that v4.x will be functionally backward compatible with prior releases, it is possible that the administrative overhead acceptable to someone with 10,000 cluster nodes may not be acceptable to someone with a 10-node cluster. While we have not yet encountered specific instances of this problem thus far, acknowledging the possibility helps to keep this issue at the forefront of the design and implementation process. It is our desire that the v4.x implementation will easily handle the small cluster case providing specific v4.0 benefits without any additional administrative overhead that may be required of the extremely large cluster installation. First is a look into v3.x followed by a brief description of issues related to v4.x development.

7.1 C3 Version 3.x

As stated above, the C3 v3.x release will provide additional features addressing the problems associated with the administration and use of clusters located in multiple domains. While the v2.x release of C3 supports multiple clusters, in order to do so, it requires that all nodes must reachable via the network – exposed to the node initiating the C3 command. This presents a problem in environments where computation nodes are hidden behind the head node by placing them on a private network with only the head node exposed to the outside network. This is easily done by using two network cards in the head node – one for the internal private network and the other for the external public network. Each network interface card (NIC) is then configured for use on the appropriate network. Another feature to make its way into v3.x is that of supporting node ranges as an integral part of the new API. This is the same use of node ranges as described in the original Ptools specification of parallel Unix commands. The final major change in v3.x is the use of the Python rather than Perl as the implementation language. While the information provided in this section should be considered preliminary, v3.x is sufficiently into its development cycle that we do not anticipate major changes from the information presented here. First, we will briefly look at the Python versus Perl issue. Second, is a discussion of multi-cluster issues and last is a description of the new API including the node range.

7.1.1 Python versus Perl

While Perl proved to be a great choice for the prior implementations of C3, it does have some flaws that Python will address. Some of the beneficial features that Perl offered include: tightly integrated regular expressions, code portability, and a fast development cycle. However, as the size of the C3 scripts increased, Perl became a liability with respect to code maintenance and feature expansion. While Python has many of the same features of Perl, it also has several features that benefit the maintainability of larger codes. For example, Perl's philosophy is that each person is different so there should be as many different ways to solve a problem. While this approach works for the smaller more personal sized scripts it makes reading another's large code implementation an effort equal to reinventing that code. In contrast, Python will typically have only one or two ways to perform a specific action thus making code more consistent between different authors. One asset of Perl is that it is well suited to text processing programs such as those needed for common system administration tasks. However, it provides some difficulty when used in large scripted programs because of this feature. While Python is capable of doing these same jobs as Perl, it is much less bound to them. For example, regular expressions are an object on Python versus Perl where they are a part of the language it self. Furthermore, Python lends itself well to the writing and maintaining of large programs. Yet another benefit is that the Python API directly supports C libraries, thus providing direct access to a much richer set of sockets and transport protocols. Because of this feature, it is possible to wrap C or C++ functions in such a manner that a programmer will never know that they are calling a C function. These calls may even throw a Python exception from C via a library call. Python, like Perl, is a freely available, open-source language that is portable and is included with many of the readily available Linux distributions. Thus, Python exhibits many of the features that make it attractive for implementing the next generation of C3 tools. For a more in-depth analysis comparing Perl and Python see [14].

7.1.2 Cluster Configuration For Multi-Cluster Environments

The original C3 cluster configuration file took the simple and most direct approach of specifying the cluster by placing one machine per line in the file. While one can cross over multiple clusters by simply placing a machine's name in the configuration file, this approach provides no method to express a cluster architecture when nodes are not directly reachable from outside the cluster. Thus, a new file format was developed in support of the v3.x multi-cluster environment. As shown in Figure 11, the cluster configuration file now consists of cluster definition blocks as follows:

```
Cluster torc{
      external_name:internal_name  #head node
      Node0                        #position 0
      Node[1-63]                   #position 1-63
      Exclude 3                    #node3 is dead
      Exclude [6-8]                #node6 - node8 are dead
      196.1.1.[30-40]              #position 64-74
      exclude 32                   #196.1.1.32 is dead
}

cluster torc2{
      head_node_name
      osiris
      dead isis                    #a dead node
      134.167.12.213
}

cluster HighTorc:head-node-name
```

Figure 11 – Multi-cluster Configuration File

A *cluster block*, as in torc and torc2, may be used to define an entire cluster to the node level. This level of definition is required on a local cluster's head node. The second technique with only the cluster name and head node provided, as shown for HighTORC, is a *cluster access alias*. This method is used to provide a loosely defined connection to another cluster. The definition is considered loose as the local C3 command will be unable to determine specific cluster attributes until it reaches the appropriate *cluster block* definition. The first cluster definition in the configuration file must be a cluster block definition and it will be assumed to be the default local cluster for commands issued from that host. This "default" will be the cluster used in the case where a C3 command is issued without explicitly specifying the cluster configuration file. When a configuration file is specified, all clusters specified within will participate in the execution of the C3 command issued. All machine names in the configuration file must be the name used to access that machine. That is, it must be the machine alias, IP address, or the fully qualified address. The cluster name is only used for internal matching purposes and is not relevant beyond the use of C3 at this time. Thus a user may specify the cluster name to be any name they wish.

In the *cluster block* definition, the first machine named in the list is always the head node. In the case where the head node has two network interface cards (NICs), with one on the external network and the other on the private internal network, the first line must contain the external name (colon separated) followed by the internal name. If there is only one network interface, the name need only be listed once (shown in Figure 11 for torc2) but can be listed as both external and internal if symmetry of the *cluster block* definitions is desired. The internal name is the one used by the computation nodes to communicate to the head node.

By default, the head node is not included in the execution of a C3 command. This is done to prevent a user from performing commands from the head node – to the head node – without explicitly desiring to do so. Without this technique, users tend to blast the head node with operations only intended for compute nodes. However, it is possible to treat the head node as a compute node and have commands execute on the head node as it does on compute nodes. This is done by listing the head node twice, first in the head node position and second as the first computation node.

The *cluster access alias* is used to specify the existence of another cluster. This technique is generally used to specify a non-local cluster where all computation nodes are only accessible via the head node on their private network. Thus, individual nodes would not be accessible to external machines even if individual node IP addresses were provided. Figure 11 shows cluster HighTORC with its externally accessible head node separated by a colon. Individual nodes will be resolved when the C3 command reaches HighTORC and accesses the local c3.conf cluster configuration file that describes itself. This scheme relieves a system administrator or user from having to synchronize their individual configuration files with those sitting on the cluster head nodes.

Four important additions to the cluster configuration file syntax are: the "#" symbol for comments, the ability to define the cluster via ranges, the `exclude` qualifier, and the `dead` qualifier. Ranges may only be specified numerically and can only be the last part of the node name. For example, instead of explicitly listing the 100 entries for `node1, node2, node3,… node100`. One can simply supply the single line consisting of `node[1-100]` for the same effect. This enables the user to define a larger cluster much more easily. The `exclude` tag indicates that the given number or range of nodes listed after the exclude tag are to be removed from the previously defined node range. Typically this is used when a group of nodes has been removed from the cluster. There is no way to override an `exclude` on the command line. Similar to the `exclude` tag is the newly added `dead` tag. Philosophically the difference is the `exclude` expects that nodes will return and `dead` tagged nodes are anticipated to be gone forever. Implementation wise, the `exclude` tag is used for ranges of nodes while the `dead` tag is reserved for single nodes.

7.1.3 Enhanced API

The original command line API for C3 was dictated by the use of the `getopt` package in Perl. This package makes parsing the command line very easy but unfortunately requires that each option be passed as a switch. One of the goals for v3.x is to make the C3 commands as similar as possible to the standard interface of the analogous Linux command. For example, in the `cps` command, the passing of options to `ps` using a `--option` or `–o` does not meet this requirement. Version 3.x implemented in Python will not use the `getopt` package and instead we will implement our own command line parser. The result is that it will be no longer necessary to preface every argument with the `--option` qualifier.

With enhanced multi-cluster support, the C3 command line API now requires a way to specify which cluster, or clusters, is targeted by a command. A cluster is designated by name (identifier) followed by a colon. Individual nodes may be specified after the colon. Likewise, a range of nodes may be specified following the colon by either indicating `node[range]` or just `[range]`. A command applies to the entire named cluster in the cluster configuration file if no node qualifiers are included on the command line. The general format of the v3.x C3 command is:

```
ccommand [OPTIONS] [CLUSTER] required options for command
```

For example, the following command will start a `ssh` daemon on the TORC cluster; on nodes 1, 4, 6 through 12, and node 14 of HighTORC; and display the machine name before executing the command:

```
        cexec –p torc: hightorc:1,4,6-12,14 /etc/rc.d/init.d/sshd start
```

Note in the above example that the cluster torc has no numbers after the colon. This indicates that the command is to execute on every node listed in the configuration file for this cluster. While on HighTORC only those nodes specified on the command line are to participate in the execution of the command. Individual nodes are indicated by their number and ranges of nodes are given by a dash-separated number range. Number refers to the relative position in the cluster specification file. Node zero (0) is the first node after the head node in the cluster definition file. Each subsequent node is assigned a number in the order listed in the file. By default the head node does not participate in a C3 command. However, a switch is provided to indicate that the operation should also occur on the head node. A node marked `exclude` or `dead` in the configuration file, but listed on the command line, will be skipped and an error message displayed. The remainder of the nodes in the command line list will still execute the indicated operation.

Two commands are provided that return the mapping of node name and number for clusters where the node names do not follow a regular pattern that is easily mapped to node|number pairs – such as cluster torc2 in Figure 11. The `cname` command returns the node name based on the numbers supplied as a command line list. The `cnum` command returns the node number based on the form of cluster_name:node_name.

The final code example shows the default cluster, the one listed first in the local cluster configuration file, executing a “`ls –l`” command on nodes 2,4,6,7,8,9,10 of the default cluster:

```
        cexec :2,4,6-10 ls -l
```

7.2 C3 v4.0

Clusters presently hold the often-fleeting position of being the most popular and powerful computing architecture today. This is an architectural paradigm where a more powerful cluster can always be built by simply adding more nodes – and this is exactly what is occurring. Thus, while the current typical computation cluster does not exceed 128-nodes, a few have expanded into the 256-node range, and it is expected that the node count will grow substantially in the near future. While it is a relatively simple matter to plug together more hardware; the difficulty is in getting the software to operate efficiently and effectively across all those nodes. To anticipate this need in scalable software, v4.x – the first “scalability” release for C3 is being developed.

To this point, the C3 tools have only been tested up to 64 nodes, where they perform rather well. We acknowledge that there are inherent limitations to the present implementation techniques used in v2.x and even v3.x of C3. Part of the problem lies in the parallel technique used in C3 itself and more serious problems come from the underlying tools used within C3. In the current parallel version of the tools, one process is created on the server to handle each client node. Within each process, the server establishes a network connection to a client. It is easily foreseeable how this methodology is severely limited when clusters of thousands of nodes are considered, as the server will be creating thousands of processes and network connections. From this example, two issues arise that need to be resolved in order to enable the C3 tools to scale. The first issue is the server’s load, which will need to be distributed among many nodes in order to prevent some of the limitations imposed by the Linux operating system, such as the maximum number of processes allowed and the maximum number of open file descriptors. The second issue is the enormous amount of network traffic that would be caused from the tools presently used by C3 in trying to service thousands of nodes. In order to reduce the amount of traffic, a more efficient means of communication to transfer data to clients must be employed.

In response to these issues, research is being done on a highly scalable version of the C3 tools. The goal of this research is to discover a means to efficiently manage and use clusters on the order of thousands of nodes. When combined with the multiple cluster functionality of C3 v3.0, the resulting tools should also be usable within the computational Grid environment, and able to automatically select the most efficient means available for an operation given the local environment. Initial research being performed includes the investigation of using various algorithms for distributing the server’s load among all cluster nodes. Work is also taking place to exploit the capability of the network hardware used in the construction of clusters to reduce the overall network traffic.

7.3 Conclusion

While clusters are the price / performance leader in high performance computing, the lack of good administration and application tools represents a hidden cost to both cluster owners and users. This paper presented the Cluster Command and Control (C3) tool suite, a number of command line tools developed as part of the Oak Ridge National Laboratory's HighTORC cluster project that are designed to reduce the cost of cluster ownership. In addition to basic command line use, examples of advanced use of the C3 tools were provided. Here, it was shown that the C3 tools could be embedded into other programs for both cluster administration and user application purposes. Furthermore, these same tools shown capable of providing the "backend" connection to administering and using the cluster via the web-based tool M3C.

Of the three criteria set forth for the C3 tools at the beginning of this work – single machine look and feel, secure, and scalable – the current version 2.7 of the C3 tools meet all expectations for clusters of reasonable size. Looking back over the past year, v1.x met all of these requirements except for that of scalability. Cluster tool scalability is an elusive target that the cluster computing community will have to continually chase as cluster node counts continue to rise. As explained in the latter part of this paper, the C3 team is in the process of addressing those issues critical to achieving a more scalable implementation of cluster tools. Of course, as tools become more scalable, computational scientists will want even larger systems. This is just part of the never-ending cycle for increased computation power.

8 REFERENCES

[1] The HighTORC system page, http://www.epm.ornl.gov/torc
[2] Rsync system documentation, http://www.rsync.samba.org
[3] OpenSSL Specification, http://www.openssl.org
[4] OpenSSH Specifications, http://www.openssh.com/
[5] ISC Dynamic Host Configuration Protocol, http:// www.isc.org/products/DHCP
[6] Systemimager documentation, http://www.systemimager.org
[7] Ptools project, http://www-unix.mcs.anl.gov/sut/
[8] The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers, Inc., San Francisco, 1999.
[9] The Open Cluster Group, http://www.OpenClusterGroup.org
[10] Open Source Cluster Application Resources (OSCAR), http://www.epm.ornl.gov/oscar
[11] Rdist home page, http://www.magnicomp.com/rdist
[12] Tridgell, A. and Mackerras, P. 1996. The rsync algorithm. Technical Report TR-CS-96-05 (June), Department of Computer Science, Australian National University.
[13] M3C Tool, http://www.epm.ornl.gov/~jens/m3ctool
[14] Open Source Developer's Journal, Perl vs Python: Which one is right for you?, p8-14, Issue No. 1.